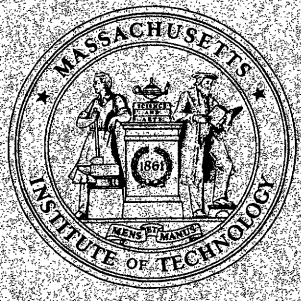
OPERATIONS RESEARCH CENTER

working paper



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
07 MAR 1990		**00-03-1990 to 00-03-1990**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Optimal Parallel Implementation of a Quadratic Transportation Algorithm	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Thinking Machines Corporation,245 First Street,Cambridge,MA,02142**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**10**	

An Optimal Parallel Implementation of a Quadratic Transportation Algorithm *

Mike McKenna
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142.

Stavros A. Zenios
Decision Sciences Department
The Wharton School
University of Pennsylvania,
Philadelphia, PA 19104.

March 7, 1990

*Research partially supported by NSF grants ECS-8718971 and CCR-8811135, AFOSR grant 89-0145 and Thinking Machines Corporation. We would like to acknowledge several useful discussions with Jill Mesirov.

Abstract

We discuss the implementation of a quadratic transportation algorithm on massively parallel computer architectures with hypercube communication networks. The implementation is optimal in the sense that it requires effectively $O(\frac{m_O m_D}{P})$ operations to perform one iteration of an $m_O \times m_D$ problem using P processors. Peak computing rates of 3 GFLOPS are achieved by the algorithm on a 64K Connection Machine CM-2.

1 Introduction

We consider in this paper quadratic optimization problems over transportation constraints. Such problems find applications in logistics, traffic management, and matrix balancing models in economics and regional planning. The dual, row-action algorithm of Zenios and Censor [1989] is well suited for implementation on massively parallel computer architectures. In this report we describe an optimal implementation of this algorithm on a massively parallel architecture with a hypercube communication network. The implementation is optimal in the sense that it requires effectively $O(\frac{m_O m_D}{P})$ operations to perform one iteration of an $m_O \times m_D$ problem using P processors. Computational experiments on a Connection Machine CM-2 indicate that the implementation achieves a computing rate of 3 GFLOPS when solving dense 1024×1024 problems on a 64K system.

2 The Quadratic Transportation Algorithm

Let $< m >$ denote the set $\{1, 2, 3, \ldots, m\}$. Denote by $\Re^m$ the m-dimensional Euclidean space. A transportation graph is defined as the triplet $\mathcal{G} = (V_O, V_D, \mathcal{E})$, where $V_O =< m_O >$, $V_D =< m_D >$ and $\mathcal{E} \subseteq \{(i,j) \mid i \in V_O, j \in V_D\}$. V_O and V_D are the sets of origin and destination nodes of cardinality m_O and m_D respectively. $\mathcal{E}$ is the set of n directed arcs (i,j), with origin node i and destination node j, which belong to the graph, $n \leq m_O m_D$. Let also $x = (x_{ij}) \in \Re^n$ be the vector of flows; $u = (u_{ij}) \in \Re^n$ be the vector of upper bounds on the flows; $s = (s_i) \in \Re^{m_O}$, for $i \in V_O$, be the vector of supplies; $d = (d_j) \in \Re^{m_D}$, for $j \in V_D$, be the vector of demands; $\pi^O = (\pi_i^O) \in \Re^{m_O}$, for $i \in V_O$, be the vector of dual prices for origin nodes; $\pi^D = (\pi_j^D) \in \Re^{m_D}$, for $j \in V_D$, be the vector of dual prices for destination nodes; $r = (r_{ij}) \in \Re^n$ be the vector of dual prices for the bound constraints; $\delta_i^+ = \{j \in V_D \mid (i,j) \in \mathcal{E}\}$ be the set of destination nodes which have arcs with origin node i, and $\delta_j^- = \{i \in V_O \mid (i,j) \in \mathcal{E}\}$, the set of origin nodes which have arcs with destination node j. With this notation we define the quadratic transportation problem as follows:

$$Minimize \quad F(x) = \sum_{(i,j)\in\mathcal{E}} \left[\frac{1}{2} w_{ij} x_{ij}^2 + c_{ij} x_{ij}\right] \tag{1}$$

$$Subject\ to:$$

$$\sum_{j\in\delta_i^+} x_{ij} = s_i, \ \forall i \in V_O, \tag{2}$$

$$\sum_{i\in\delta_j^-} x_{ij} = d_j, \ \forall j \in V_D, \tag{3}$$

$$0 \leq x_{ij} \leq u_{ij}, \ \forall (i,j) \in \mathcal{E}. \tag{4}$$

where $\{w_{ij}\}$ and $\{c_{ij}\}$ are given positive real numbers. The following dual, row-action, algorithm for this problem was developed in Zenios and Censor [1989], where its asymptotic convergence was established.

Step 0: (Initialization) Set $k \leftarrow 0$. Get $x^0, (\pi^O)^0, (\pi^D)^0, r^0$ such that:

$$x_{ij}^0 = -\frac{1}{w_{ij}}\left[c_{ij} + (\pi_i^O)^0 + (\pi_j^D)^0 + r_{ij}^0\right]. \tag{5}$$

Step 1: (Iterative step over constraint set (2)).

$$\rho_i^k = \frac{1}{\sum_{j\in\delta_i^+} 1/w_{ij}}\left[s_i - \sum_{j\in\delta_i^+} x_{ij}^k\right], \tag{6}$$

$$x_{ij}^k \leftarrow x_{ij}^k + \frac{1}{w_{ij}}\rho_i^k, \quad j \in \delta_i^+, \tag{7}$$

$$(\pi_i^O)^{k+1} = (\pi_i^O)^k - \rho_i^k. \tag{8}$$

Step 2: (Iterative step over constraint set (3)).

$$\sigma_j^k = \frac{1}{\sum_{i\in\delta_j^-} 1/w_{ij}}\left[d_j - \sum_{i\in\delta_j^-} x_{ij}^k\right], \tag{9}$$

$$x_{ij}^k \leftarrow x_{ij}^k + \frac{\sigma_j^k}{w_{ij}}, \quad i \in \delta_j^-, \tag{10}$$

$$(\pi_j^D)^{k+1} = (\pi_j^D)^k - \sigma_j^k. \tag{11}$$

Step 3: (Iterative step over constraint set (4)).

$$\Delta_{ij}^k \doteq \text{mid}\,\{r_{ij}^k, w_{ij}(u_{ij} - x_{ij}^k), -w_{ij}x_{ij}^k\}, \tag{12}$$

$$x_{ij}^{k+1} = x_{ij}^k + \frac{\Delta_{ij}^k}{w_{ij}}, \tag{13}$$

$$r_{ij}^{k+1} = r_{ij}^k - \Delta_{ij}^k. \tag{14}$$

Step 4: Replace $k \leftarrow k+1$ and return to Step 1.

3 An Optimal Implementation on the Connection Machine

The key to the implementation of the algorithm on the Connection Machine CM–2 hypercube is the configuration of the processing elements into a two-dimensional grid. If the grid size is larger than the number of processing elements then we configure the machine into an $m_O \times m_D$ grid of *virtual processors* (VP), and each physical processing element performs the work of several virtual processors. The dimension of the grid is set equal to the number of origin nodes m_O rounded up to the nearest integer that is a power of two,

and the other dimension is set equal to the number of destination nodes m_D rounded up in the same fashion. Virtual processors outside the $m_O \times m_D$ grid are disabled and do not participate in the computations. The memory of virtual processor with grid coordinates (i, j) is partitioned into the following data fields: (1) Supply and demand, s and d, (2) dual prices, π^O, π^D and r, (3) Upper bound u, (The lower bound l is assumed to be equal to zero and hence it is treated as a constant.), (4) current iterate x, (5) a scaling factor *scale* used to store ρ, σ or intermediate results, and (6) two fields IW and OW hold the terms $\frac{1}{\sum_{i \in \delta_j^-} 1/w_{ij}}$ and $\frac{1}{\sum_{j \in \delta_i^+} 1/w_{ij}}$ respectively. The algorithm is executed as follows:

Step 1: A spread-sum operation along the 1-axis of the grid computes the partial sums $\sum_{j \in \delta_i^+} x_{ij}^k$ for each origin node (i.e., each row of the grid). This result is spread to the *scale* memory fields of all VP in the same row. A sub-mult operation (i.e., an optimized implementation of $a(x - b)$) is used to compute the scaling factor ρ (equation (6)). The scaling factor updates the local field x by addition and the local field π^0 by subtraction (equations (7) and (8) respectively).

Step 2: This step is similar to Step 1, with a spread-sum along the 0-axis.

Step 3: A combination of min and max operations computes the $mid(.)$ of equation (9), which is then used to update the local field x by a mult-add operation (equation (11)) and local field r (equation (11)).

3.1 The Hypercube Implementation

In this section we describe how Steps 1-3 of the algorithm are implemented to run on P physical processors of the CM–2 hypercube, so that the execution time is in effect $O(\frac{m_O m_D}{P})$. To simplify our discussion, assume that m_O, m_D and $\sqrt{P}$ are powers of two. Let the vertices of the hypercube be labeled with the integers 0 through $P - 1$, where vertices i and j of the cube are connected with a wire when the absolute value of $i - j$ is a power of two. We initially configure the $m_O \times m_D$ grid of virtual processors so that each physical processor is assigned to do the work for an $\frac{m_O}{\sqrt{P}} \times \frac{m_D}{\sqrt{P}}$ subgrid of virtual processors. In figure 1 we have $m_O = 32$, $m_D = 32$, $P = 16$ and each physical processor does the work for an 8×8 subgrid of virtual processors.

In our configuration, we lay out the physical processors in a $\sqrt{P} \times \sqrt{P}$ two-dimensional grid. Let P_{ij} be the the physical processor that is in row i and column j of the grid. We place physical processor P_{ij} at vertex $i\sqrt{P} + j$ of the cube. In the example of figure 1, the 16 physical processors are configured in a 4 by 4 grid, and each physical processor is placed at vertex $4i + j$ of the hypercube. The arcs in figure 2 illustrate how the CM–2 hypercube connects the $\sqrt{P} \times \sqrt{P}$ grid of physical processors.

Most of the operations in Steps 1-3 require no communication between virtual processors. For these operations, each virtual processor spends O(1) time to work on its local memory, and each physical processor performs the work of $\frac{m_O m_D}{P}$ virtual processors. Therefore the local operations require $O(\frac{m_O m_D}{P})$ time.

The operations that do require communications are the spread-sums, where we add the elements in each row of the virtual grid, and copy the results back to every processor of each virtual row. Technically, a spread-sum is an operation where a field $scale_{ij}$ is allocated in each virtual processor, and $scale_{ij} = \sum_{k=0}^{m_D} x_{ik}$. If the transportation graph is not complete, then for each missing edge (i,j), x_{ij} is set to zero before performing the spread-sum. In the rest of this section, we show how a spread-sum is performed, effectively, in $O(\frac{m_O m_D}{P})$ time.

Let $x_{ij,ab}$ denote the value of x that lies at position (a,b) in the subgrid of physical processor P_{ij}. In figure 1, physical processor P_{ij} holds the values

$$\begin{array}{ccccc} x_{ij,00}, & x_{ij,01}, & x_{ij,02}, & \cdots & x_{ij,07}, \\ x_{ij,10}, & x_{ij,11}, & x_{ij,12}, & \cdots & x_{ij,27}, \\ \vdots & & & & \vdots \\ x_{ij,70}, & x_{ij,71}, & x_{ij,72}, & \cdots & x_{ij,77} \end{array}$$

A spread-sum is performed in three phases. In the first phase, each physical processor finds the subtotal for each row in its subgrid. For example, in figure 1, each physical processor performs the following 8 additions:

$$\begin{array}{ccccccccccc} x_{ij,00} & = & x_{ij,00} & + & x_{ij,01} & + & x_{ij,02} & + & \cdots & + & x_{ij,07} \\ x_{ij,10} & = & x_{ij,10} & + & x_{ij,11} & + & x_{ij,12} & + & \cdots & + & x_{ij,17} \\ \vdots & & \vdots & & & & & & & & \vdots \\ x_{ij,70} & = & x_{ij,70} & + & x_{ij,71} & + & x_{ij,72} & + & \cdots & + & x_{ij,77} \end{array}$$

Once these additions have been performed, each physical processor holds a column vector of subtotals (the shaded elements in figure 1). Obtaining these subtotals requires $O(\frac{m_O m_D}{P})$ time.

Let C_{ij} denote the column of subtotals that lies in physical processor P_{ij}. In the second phase of the spread-sum, the processors in each physical row i coordinate to perform efficiently the vector sum $SUM_i = C_{i0} + C_{i1} + \cdots + C_{i(\frac{m_D}{\sqrt{P}}-1)}$. The resulting vector SUM_i holds the spread-sums for the rows of the virtual grid that are assigned to physical row i. In figure 1, each vector SUM_i would hold the spread-sums for virtual rows $8i$ through $8i+7$.

Now note that the physical processors P_{i0}, P_{i1}, P_{i2}, ... of physical row i all lie in a physical subcube of the machine that spans $\log_2\sqrt{P}$ dimensions. (In figure 2, the thick arcs show how each row of the physical grid spans a two-dimensional subcube.) Let $C_{ij,k}$ denote the kth element in column vector C_{ij}, and let $SUM_{i,k}$ denote the kth element in column vector SUM_i. Given the $\log_2\sqrt{P}$-dimensional subcube, we use the procedure listed below to obtain the first $\log_2\sqrt{P}$ elements of vector SUM_i. (Note that the CM–2 hypercube wires are bi-directional and can be activated simultaneously.)

Let $\ell = \log_2\sqrt{P}$;
For $k = 0$ through $\ell - 1$ **do**
(S1) Let each processor P_{ij} send
$C_{ij,0}$ along dimension $(k+0)$ mod ℓ of the subcube,
$C_{ij,1}$ along dimension $(k+1)$ mod ℓ of the subcube,
$\vdots$
$C_{ij,\ell-1}$ along dimension $(k+\ell-1)$ mod ℓ of the subcube;
(S2) Let each processor P_{ij} receive
a value $temp_0$ along dimension $(k+0)$ mod ℓ of the subcube,
a value $temp_1$ along dimension $(k+1)$ mod ℓ of the subcube,
$\vdots$
a value $temp_{\ell-1}$ along dimension $(k+\ell-1)$ mod ℓ of the subcube;
(S3) **For** $q = 0$ through $\ell - 1$ **do**
Let $C_{ij,q} = C_{ij,q} + temp_q$;

Problem	Itns.	VP ratio	Real time (seconds)	CM time (seconds)	Real GFLOPS (64K CM-2)	CM GFLOPS (64K CM-2)
TEST1	5000	256	556.49	532.92	3.012	3.148
TEST2	4600	128	270.24	255.62	2.856	3.019
TEST2	4000	256	730.00	676.00	1.839	1.986

Table 1: Performance of the quadratic transportation algorithm using 1024 x 1024. First and second rows report results with the optimal implementation of the algorithm. Third row reports results using C/Paris instructions from the CM-2 library, release 5.2.

At the end of this procedure, each value $C_{ij,k}$ for $k = 0, ..., \ell - 1$ equals $SUM_{i,k}$. If we repeat this procedure for consecutive groups of ℓ elements along vector C_{ij}, then eventually each vector C_{ij} equals SUM_i, which completes phase two of the spread-sum.

Each execution of steps (S1) and (S2) requires O(1) time. On a Connection Machine, the execution time for the loop in step (S3) is negligible when compared to the communication time in steps (S1) and (S2); so we treat the loop in step (S3) as an O(1) operation. The outer loop of this procedure iterates ℓ times; so the whole procedure requires $O(\ell)$ time to execute. The procedure itself is repeated for $\frac{m_Q}{\ell\sqrt{P}}$ groups of ℓ elements; so phase two of the spread sum requires $O(\frac{m_Q}{\sqrt{P}})$ time.

If step (S3) of the above procedure was not dominated by steps (S1) and (S2), then we would have to say that phase two of the spread-sum requires $O(\frac{m_Q}{\sqrt{P}}\log_2\sqrt{P})$ time. In that case, we could remove the $\log_2\sqrt{P}$ factor by using a straightforward variant of the more complex one-to-all broadcast algorithm that is given by Johnsson and Ho [1989].

In phase three of the spread-sum, we copy the column vector C_{ij} (which now equals SUM_i) across processor P_{ij}'s subgrid, so that each virtual processor holds the result of a spread-sum. This phase requires $O(\frac{m_Q m_D}{P})$ time. Adding the execution times for the three phases of the spread-sum, we get an execution time that is effectively $O(\frac{m_Q m_D}{P})$.

3.2 Numerical Results

We implemented the algorithm of Section 2 as explained in Section 3 in C/Paris, using CMIS instructions, on a CM-2 with 32-bit floating point accelerators. The performance of the parallel implementation was evaluated using two randomly generated test problems of dimension 1024 x 1024. The implementation uses virtual processors at a ratio of 256 and 128 virtual processors per physical processor, on a 4K and 8K CM-2 respectively. The computing rate of the algorithm under different virtual processing ratios is summarized in Table 1. The last row of the same table provides, as a benchmark, the performance of a C/Paris implementation of the algorithm without using the techniques of section 3. Observe that the computing rate estimated based on CM time is consistently in excess of 3 GFLOPS. Computing rates estimated based on real time exceed 3 GFLOPS for higher virtual processing ratios. This discrepancy was anticipated since the first phase of spread-sum requires no communications, while hypercube communications are needed in the second phase of the operation. The C/Paris implementation is significantly slower than the optimal implementation. The difference in the number of iterations between the two runs for problem TEST2 is due to the difference in the terminal tolerance specified. Both implementations achieve identical final error in exactly the same number of iterations.

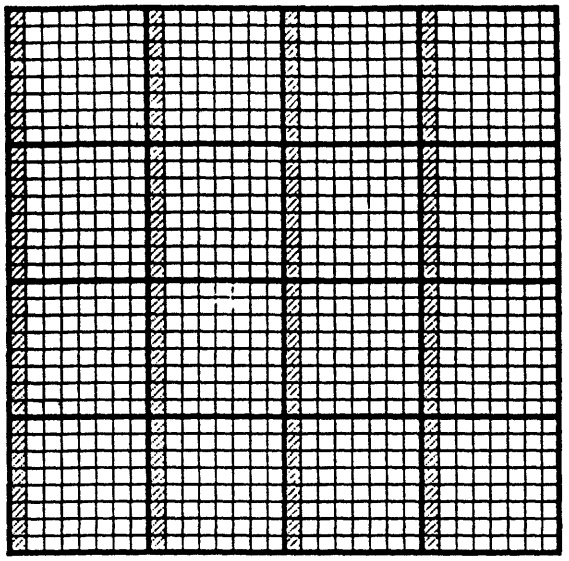


Figure 1: Virtual Processor Configuration.

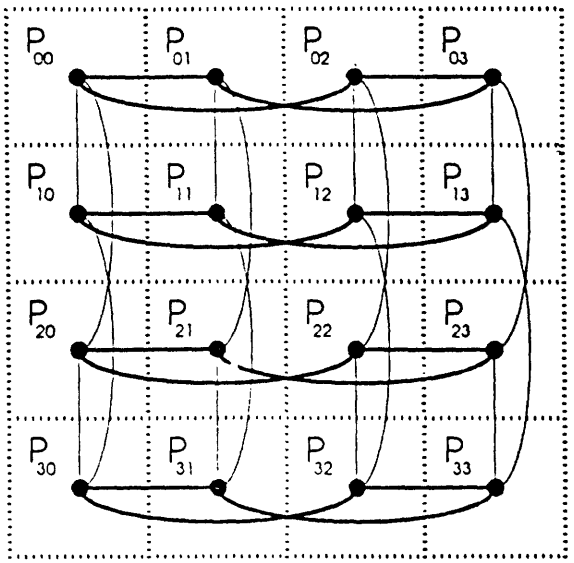


Figure 2: Physical Processor Configuration.

4 Conclusions

Gigaflop performance has been quite common in several areas of large scale scientific computing using massively (and other) parallel architectures. Unfortunately such performance is difficult to achieve in numerical optimization. Optimization problems do not usually have nice structures at a micro level, and optimization algorithms need frequent communications. We have shown that for some well structured problems we may overcome the communication bottleneck and achieve performance in the range of 3 GFLOPS. An implementation that is effectively optimal was achieved using a simpler scheme than the one of Johnsson and Ho [1989].

References

[1] S.L. Johnsson and C.T. Ho., "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Transactions on Computers*, vol. 38, no. 9, Sept. 1989, p. 1249-1268.

[2] S.A. Zenios and Y. Censor. *Massively Parallel Row-Action Algorithms for Some Nonlinear Transportation Problems.* Report 89–09–10, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1989.